仮想記憶機構とプログラミング (数値計算のアルゴリズムの研究)

AUTHOR(S):

石田, 晴久

CITATION:

石田, 晴久. 仮想記憶機構とプログラミング (数値計算のアルゴリズムの研究). 数理解析研究所講究録 1974, 199: 28-48

ISSUE DATE:

1974-01

URL:

http://hdl.handle.net/2433/107326

数理解析研究所講究録
199巻 1974年 28-48

仮想記憶機構とプログラミング

東京大学大型計算機センター
石田 晴久

1. はじめに

最近いろいろな計算機（HITAC 8800/8700, IBM 370シリーズ, FACOM 230の8シリーズなど）に採用されるようになってきた仮想記憶は広義にとれば，バッファー・メモリ，主記憶，ドラム（またはディスク）などからなるメモリのハイラーキー構造をユーザから隠し，あたかも非常に大きなメモリがあるかのごとくみせかける技術である．東大のHITAC 8800/8700システムの例をとれば，このメモリ・ハイラーキーは図1のようになっている．

こうした現実のメモリー・ハイラーキーは仮想記憶では（プログラマの意識する論理アドレスと計算機が使う実アドレスとが完全に分離されているため）プログラマは意識しなくてもよいことになっているが，実際にはやはりそれを意識してプログラムを書く方が処理効率は高くなることが多い．これ

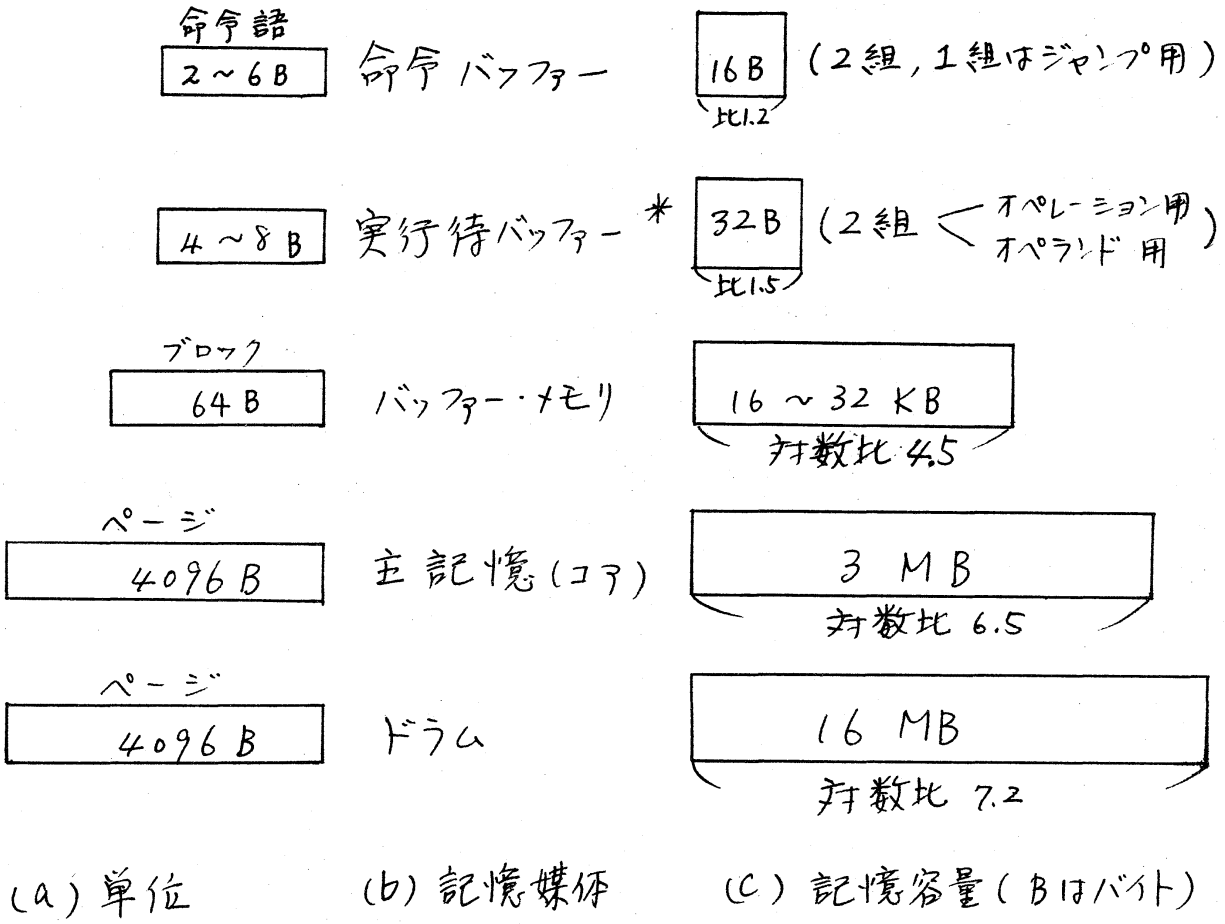


図1. メモリー・ハイラーキーの例 (HITAC 8800/8700)

[*他に汎用レジスタ16個(各4B)，浮動小数実数レジスタ4組(各8B)]

はとくに大きなプログラムや大きなデータを扱うときにそうであって，現在の技術では大きなメモリを使うにはそれなりの工夫がいることを意味している．

ところで仮想メモリ系でメモリ・ハイラーキーを意識するとはいっても，単にプログラムを書く立場からいえば，仮想記憶の制御機構をそう細かに知る必要はない．最も知る必要のあるのは，プログラムが主記憶内のある与えられた領域（これを主記憶枠と呼ぶ）に入りきらないとき，残りの部分はドラムに置かれ，その後必要に応じて主記憶とドラムの間で情報の転送が行われるが，そのときどんなアルゴリズムでこの転送が行われるかということである．さらにもう一段下のレベルではバッファー・メモリのサイズを意識するとよいであろう．

本稿ではHITAC 8800/8700 システムを例にとって，仮想記憶を中心にメモリー・ハイラーキーの制御機構を簡単に紹介し，それとプログラミングとの関係を考察する．

2. 仮想記憶とスワッピング・アルゴリズム

HITAC 8800/8700 ではIBM 370 と同様メモリは4キロバイト（KB）＝1キロ語（KW）のページという単位に分れて

いる．主記憶（コア）とドラムとの間で情報が転送されるとき，転送の単位となるのがこのページである．次にこの転送の行われ方をHITAC8800/8700のオペレーティング・システム（OS7）におけるメモリ制御にもとづいて説明する．

図2に示すように，OS7では実行可能な形になったオブジェクト・プログラムはいったんディスクに用意され，その後OSにより主記憶およびドラム内にメモリ領域の割当てを受けて実行されてゆくことになる．この際，主記憶は多重プログラミングで同時に走る他のプログラム（ジョブあるいはタスク）との共同利用になるから，ひとつのプログラムでその全体が使えるわけではなく，OSによって割当てられたある領域に限られる．この領域が前述の主記憶枠である．

もっと具体的にいえば，東大センターのシステムでは，ユーザの指定できるジョブ・クラス毎に，主記憶枠の標準値を次のように決めている．

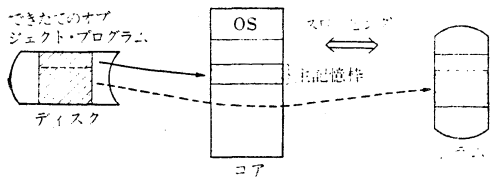


図2 プログラムのローディング

ジョブクラス	CPU時間の上限	プログラムの大きさの上限	主記憶枠
A	10秒	60 KW	34 KW
B	1分	320 KW	60 KW
C	10分	320 KW	80 KW
D	60分	1500 KW	256 KW (最大600 KW)

そこで，たとえばジョブ・クラスCで320 KW のメモリを使うプログラムがあったとすると，それが実行される際，図1でその4分の1の80KWだけ*が主記憶にはいり，残りの240KWはドラムに置かれることになる。

こうした主記憶とドラムとの間のメモリの割当て制御は，主記憶とドラムとの間の転送（スワッピング）を含めて，OS7では次のように行なわれる。

(1) プログラムが割当てられた主記憶枠より小さいときは，そこにスッポリはいりきるので，その中でそのまま実行

＊ ただしそのとき主記憶がたまたまあいていれば，もっと大きな主記憶枠がとられることもある。またプログラムが主記憶枠上限より小さければプログラムの大きさ分の主記憶枠がとられる。OS7ではこの意味で主記憶枠はプログラム実行中にダイナミックに変化する。

される。スワッピングはもうろん起らない。

(2) プログラムが主記憶枠より大きいときは，プログラムの最初の部分から主記憶内へ入れ，はみ出した部分はドラムへ入れる。このときのドラムへの転送をスワップアウトという。

(3) プログラムの実行が始まって，命令やデータの取出しのために，アドレスの参照が起るたびに，システム内で論理アドレスから実アドレスへの変換が行なわれる（実はこの変換をどのようにして行なうかが，仮想メモリの中心的課題であるが，ここではふれない）。この実アドレスがもとめられる過程で，そのアドレスの含む情報が主記憶にないことが判明すると割込みが起る。

(4) この割込みが起こると，ドラムからいま必要な情報を転送してくることになる。このドラムから主記憶への転送をスワップインとよぶ。OSではこの転送の単位は1ページ（1KW＝4KB）である。そこにおめる情報のはいっているページをドラムからもってくることになるが，それには主記憶枠のなかのどれか1ページを追い出さなければならない。しかしこの追い出しの際，そのページにまったく変更（書込み）がなく，しかもそれのコピーがドラムに残っていれば，そのページは改めてドラムに

転送する必要はない。このため主記憶内の各ページごとにそのページに対して変更が行なわれたかどうかを表わす1ビットの目印が主記憶内[*]に必要になる。これをCビットという。そのCビットは初めOでそのページへの書込みが起こるとハードウェアでC=1にするわけである。さてページを追い出すにあたって重要なのは，どのページを追い出すかである。この決め方をスワッピング・アルゴリズムあるいは置換アルゴリズムという。OSで使われているのはLRU (Least Recently Used) アルゴリズムを簡単化したFINUFO (First In Not Used First Out) アルゴリズムである。この面白い名のアルゴリズムは，要するに"初めにはいってまて使われていないページからまず追い出す"という意味である。そこでこれを実現させる為に，各ページごとにそのページが参照されたかどうかを表わす1ビットの目印を主記憶内[**]に設けておく。これをRビットとよび，R=0は最近は参照されていない状態を表わし，そのページ内のメモリ語が参照されると，ハードウェアでR=1にするものとしておく。ここでいう参照とは，読出し・書込み・実行のいずれかである。

** 正確にいえば主記憶内のストレージ・キーはRビット，Cビットおよび3ビットのキーよりなる。

さて各主記憶枠内で常駐ページ（制御スタック領域など）を除いた追い出し可能な各ページは図3のように連結リスト構造になっている。このページ・リストにはポインタがひとつあり，それが常にどれかのページをさしている。追い出すページを決めてスワップを行うためのFINUFOアルゴリズムによる手順は次のとおりである。

(a) まずポインタのさしているページのRビットを調べる。*
R=0なら最近参照されていない証拠だから，そのページを追出す。R=1ならこれをR=0にかえて，ポインタを次のページをさす位置に進めて，同じことを繰返す。

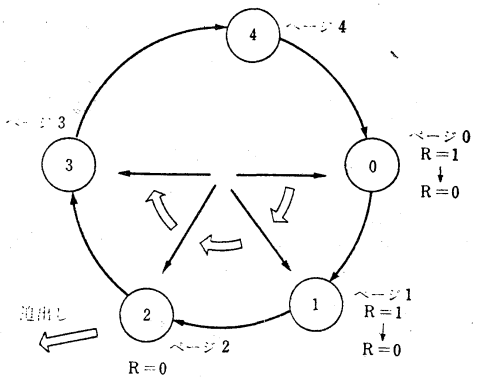


図3 FINUFOアルゴリズム

---

* これにはTRSK（Test and Reset Storage Key）命令が使われる。

(b) 追い出すべきページが決まったら，そのページのCビットを調べる*。C=0なら変更がなかったしるしだから（しかもドラムには当のページのコピーがある通常の場合は），そのページのドラムへの転送はしないが，C=1ならこのページをドラムへ転送（スワップアウト）する。

(c) 次にドラムから新たに必要とするページをいまあいたページへ転送（スワップイン）し，C=R=0にする**。またポインタは次のページへ進めておく。

このようにすれば，プログラム全体に比べて小さな主記憶群を使って，適当にスワッピングを起こさせながらプログラムを実行させることができるわけである。なお実際には，多重の多重プログラミングのもとでは，どのユーザ空間あるいはシステム空間からページを追い出すかの選択がある。これはOS7では，システム空間および各ユーザごとに単位時間当りのスワップ回数を常時測定しておき次のようにしている。

(a) システム空間の追い出し可能ページ数が大きく，スワップ回数が少ない場合には，システム空間から追い出す。

---

* これにはISK(Insert Storage Key)命令が使われる

** これにはSSK(Set Storage Key)命令が使われる

(B) n個のユーザ空間のいずれかから追い出すときは，主記憶枠以上にメモリを使っている割にスワップ回数が少ないユーザの空間から追い出す。

この場合実行するジョブの多重度が大すぎると，スワップ回数ばかり増えて有効な仕事ができなくなる。これをスラッシング（thrashing，原義はバタバタ打つこと）とよぶが，OSではこの状態が検出されると，自動的に多重度を減らすような制御が行なわれている。

3．仮想記憶におけるプログラミング上の注意

前述のスワッピング・アルゴリズムから明らかなように，仮想記憶上で走るプログラムを書く際には，プログラム走行中の各時点で必要とされるメモリ量（これをworking setと呼ぶ）を少くするよう心掛けるとよい．ワーキング・セットの小さいプログラムはまた局所性（locality）が高いともいう．一方バッファー・メモリおよび仮想記憶両方について，データの読出しに比べて書込みの方はずっと遅いので，書込み（代入）もなるべくしない方がよい．これらを具体的にコーディング上の注意としてまとめてみると次のようになる．

(1) プログラムは小さなループをまわりながら少しずつ先へ

進むように構成する.

(2) 中間結果の書込み(代入)を避ける. あるいは中間変数の個数をへらす.

(3) 配列はメモリ内での格納順にアクセスする. また同じ配列へのアクセスを繰返すときは往復する形でアクセスする.

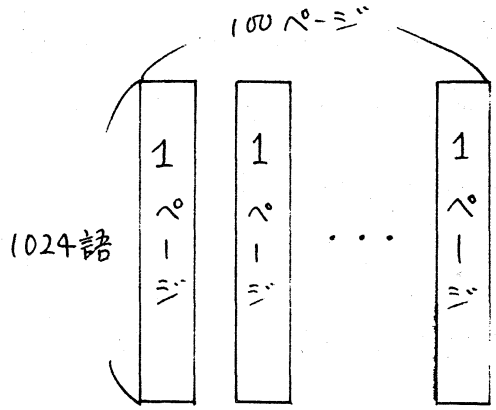
とくにFORTRANでDIMENSION A(1024, 100)のとき

```
      DO  100  J=1,100
      DO  100  I=1,1024
100   A(I,J)=0.0
```

とするのはよい(図4(b))が, (主記憶枠が100kW以下で最適化の指定をしないとき)逆に

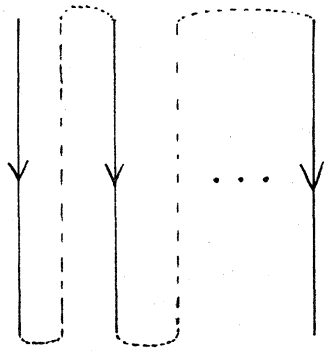
```
      DO  100  I=1,1024
      DO  100  J=1,100
100   A(I,J)=0.0
```

としたら大変である. これでは図4(c)のように格納順序とは逆の方向にアクセスすることになり, 仮に主記憶枠内でこの配列の入る部分が50kWしかないとすると, 図4(d)のように当の主記憶枠の内容はわずか50語アクセスする毎にすっかり入れ替ってしまう. つまり1ページスワップしては1語アクセスするだけとなり, これでは計算機は全くのドラム・コンピュータになってしまうわけである.



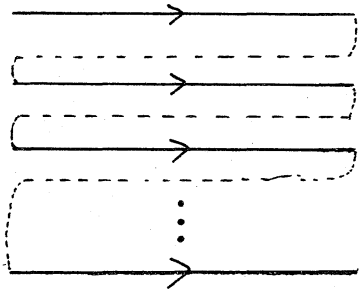
(a) 配列の格納

列方向に順に入る



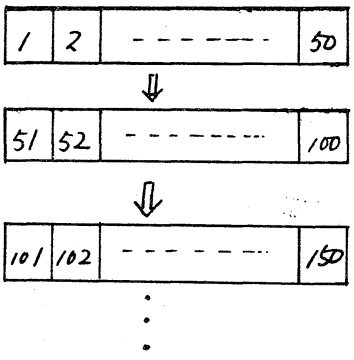
(b) 順方向アクセス

1ページ内の1024語を順にアクセスして次のページへ



(c) 逆方向アクセス

1ページ内で1語ずつアクセスして次のページへ



(D) 逆方向アクセスのときの主記憶部の内容の変化

各升目は1ページ(1KW)を表わす。番号はTに対応

図4 配列のアクセス法

(4) ひとつのループ内で使う変数・配列・副プログラムなどは連続したメモリ領域にまとめて格納されるようにする．この意味でDIMENSION文に書く変数や配列の順序には注意した方がよい．プログラム単位の順番もそうである．

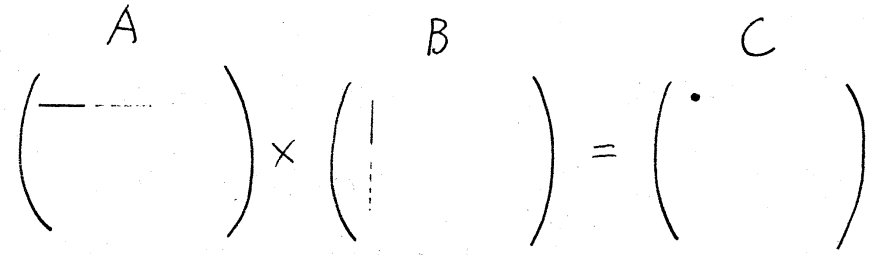
(5) 大きな配列は小さな部分に分割して一部分ずつアクセスする．

4．バッファー・メモリとプログラミング

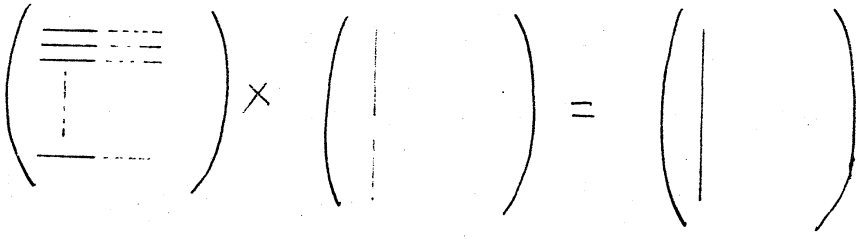
仮想記憶向きのプログラムを作った場合，あるいはプログラムやデータが比較的小さい場合，次に問題になるのはバッファー・メモリの効果である．

図5にバッファー・メモリのサイズ（HITAC 8800で8kW，HITAC 8700で4kW）を意識した計算法の1例を示す．このやり方で一度にアクセスする配列要素の数をlとすると，各段階でのデータ領域の広さは次のようになる．

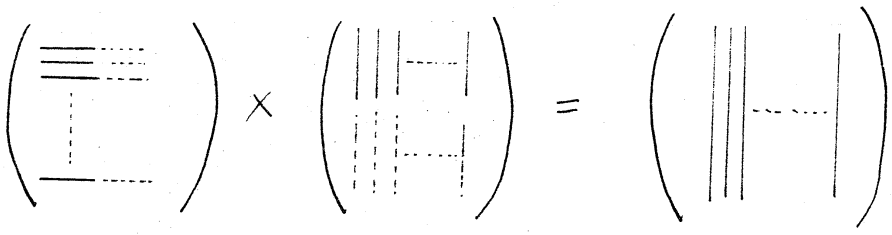
	データ数	$N=100, l=20$	$N=200, l=20$	$N=300, l=20$
第1段階	$2l+1$	41	41	41
第2段階	$N(l+1)+l$	2,120	4,220	6,320
第3段階	$N(N+2l)$	14,100	48,000	92,000
従来の方法	$3N \times N$	30,000	120,000	270,000



(a) 第1段階　A_{11} ～ A_{i1} と B_{11} ～ B_{i1} をかけて C_{11} の部分積を出す（実線部）



(b) 第2段階　Aの列を1からNまで変えて C_{j1} の部分積を出す



(c) 第3段階　Bの行を1からNまで変えていってC全体の部分積を出す

(d) 以下Aの次の部分とBの次の部分（点線部）について同様なことを繰返す

(e) 残りの部分についても同様に繰返す。

図5　バッファー・メモリ向きの計算法

図6はこの部分積による方法と従来の方法とのCPU時間比較を行うプログラムと測定された時間の例である．ここではメモリは計131 kW，主記憶枠60 kWであるが，主記憶がすいていて主記憶枠が拡げられたらしく，スワッピングは計47回しか起こらなかった．スワッピングがもっと起っていれば時間の差はもう少し開いたものと思われる．*

次にこのような行列をひとつのループ内で数多く使うときには，バッファー・メモリがセクタ（HITAC 8800で2kB=512w，HITAC 8700で1kB=256w）という単位に分れていて，しかもそうしたセクタが16個しかないという事実が利いてくることがある．こうしたセクタは通常プログラム（純手続き）部分，定数・変数部分，アドレス定数部分，作業エリアに各1個ずつ使われることが多いので，行列を入れる分は12セクタ程度となる．そこで各行列の大きさが1セクタ分以上あるいは各行列の格納場所が1セクタ以上離れている場合，各行列は別々のセクタに入ることになるから，ひとつのループ内に13個以上の行列が出てくると，全部はバッファー・メモリに入りきらなくなる．図7のようなプログラムを試してみると，この効果は図8のようにはっきり現われる．

---

*ジョブ・クラスIを使い緊急タスクとしてHITAC 8800のみで処理させた．

```
      INTEGER A(2  ,2  ),B(2  ,2  ),C(2  ,2  )
      L=2
      DO 500 N=L,200,L
      DO 1   J=1,N
      DO 1   I=1,N
      A(I,J)=I/1
      B(I,J)=J/1
      C(I,J)=0
    1 CONTINUE
      CALL CLOCK(IC1,3)
      DO 3   K=1,N
      DO 3   I=1,N
      DO 3   J=1,N
    3 C(I,K)=A(I,J)*B(J,K)+C(I,K)
      CALL CLOCK(IC2,3)
      C1=(IC2-IC1)/10.
      DO 4   J=1,N
      DO 4   I=1,N
    4 C(I,J)=
      CALL CLOCK(IC1,3)
      DO 200 M=1,N,L
      MM=M+L-1
      DO 2   K=1,N
      DO 2   I=1,N
      DO 200 J=M,MM
  200 C(I,K)=A(I,J)*B(J,K)+C(I,K)
      CALL CLOCK(IC2,3)
      C2=(IC2-IC1)/10.0
      WRITE(6,601)C1,C2,N
  601 FORMAT(1H ,2F15.3,I10)
  500 CONTINUE
      STOP
      END
```

普通の
求積法

部分積
による
方法

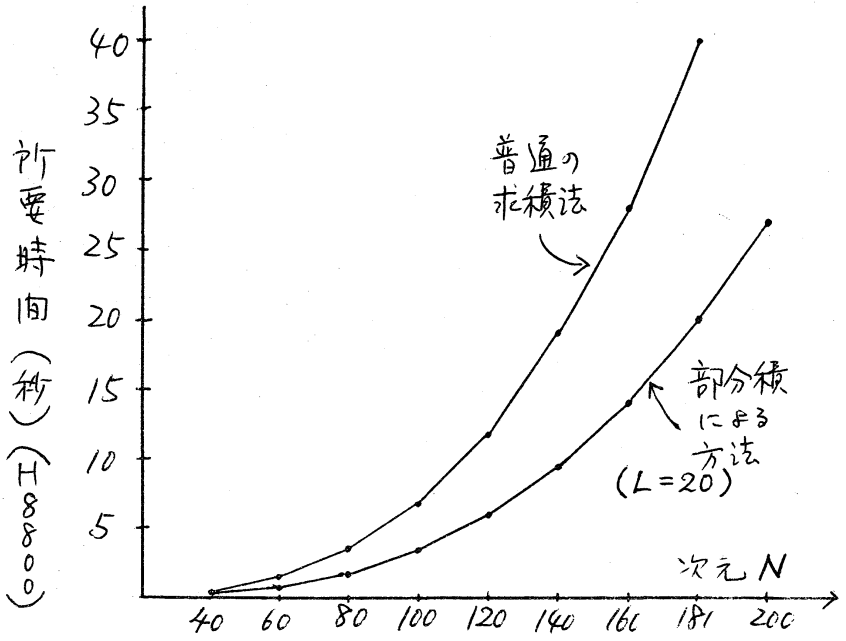


図6　行列の積を求めるプログラム

```
      DIMENSION DUMMY(903)
      DIMENSION A1(1024),A2(1024),A3(1024),A4(1024),A5(1024)
      DIMENSION A6(1024),A7(1024),A8(1024),A9(1024),A10(1024)
      DIMENSION A11(1024),A12(1024),A13(1024),A14(1024),A15(1024)
      DIMENSION A16(1024),A17(1024),A18(1024),A19(1024),A20(1024)
      DIMENSION A21(1024),A22(1024),A23(1024),A24(1024),A25(1024)
      DIMENSION A26(1024)
      DO 300 L=1,13
      CALL CLOCK(IC,3)
      DO 100 J=1,10
      DO 100 I=1,1024
      A1(I)=A2(I)
      IF(L.LE.1) GO TO 200
      A3(I)=A4(I)
      IF(L.LE.2) GO TO 200
      A5(I)=A6(I)
      IF(L.LE.3) GO TO 200
      A7(I)=A8(I)
      IF(L.LE.4) GO TO 200
      A9(I)=A10(I)
      IF(L.LE.5) GO TO 200
      A11(I)=A12(I)
      IF(L.LE.6) GO TO 200
      A13(I)=A14(I)
      IF(L.LE.7) GO TO 200
      A15(I)=A16(I)
      IF(L.LE.8) GO TO 200
      A17(I)=A18(I)
      IF(L.LE.9) GO TO 200
      A19(I)=A20(I)
      IF(L.LE.10) GO TO 200
      A21(I)=A22(I)
      IF(L.LE.11) GO TO 200
      A23(I)=A24(I)
      IF(L.LE.12) GO TO 200
      A25(I)=A26(I)
200   CONTINUE
100   CONTINUE
      CALL CLOCK(IA,3)
      TIME=(IA-IC)/10.0
      WRITE(6,600) L,TIME
600   FORMAT(1H ,I5,5X,F10.1)
300   CONTINUE
      STOP
      END
```

SYMBOL	TYPE	LOCATION (16進)
A1	AR 4	00001000
A6	AR 4	00006000
A11	AR 4	0000B000
A16	AR 4	00010000
A21	AR 4	00015000
A26	AR 4	0001A000

DUMMYを入れたので各配列は16進で1000バイト，すなわち10進で4kB(ページ)の区切りから始まっている．

(答)	(msec)
1	105.9
2	129.8
3	153.9
4	177.4
5	202.0
6	226.6
7	753.8
8	1085.4
9	1305.2
10	1530.6
11	1650.2
12	1803.5
13	2144.3

図7　配列の代入プログラム

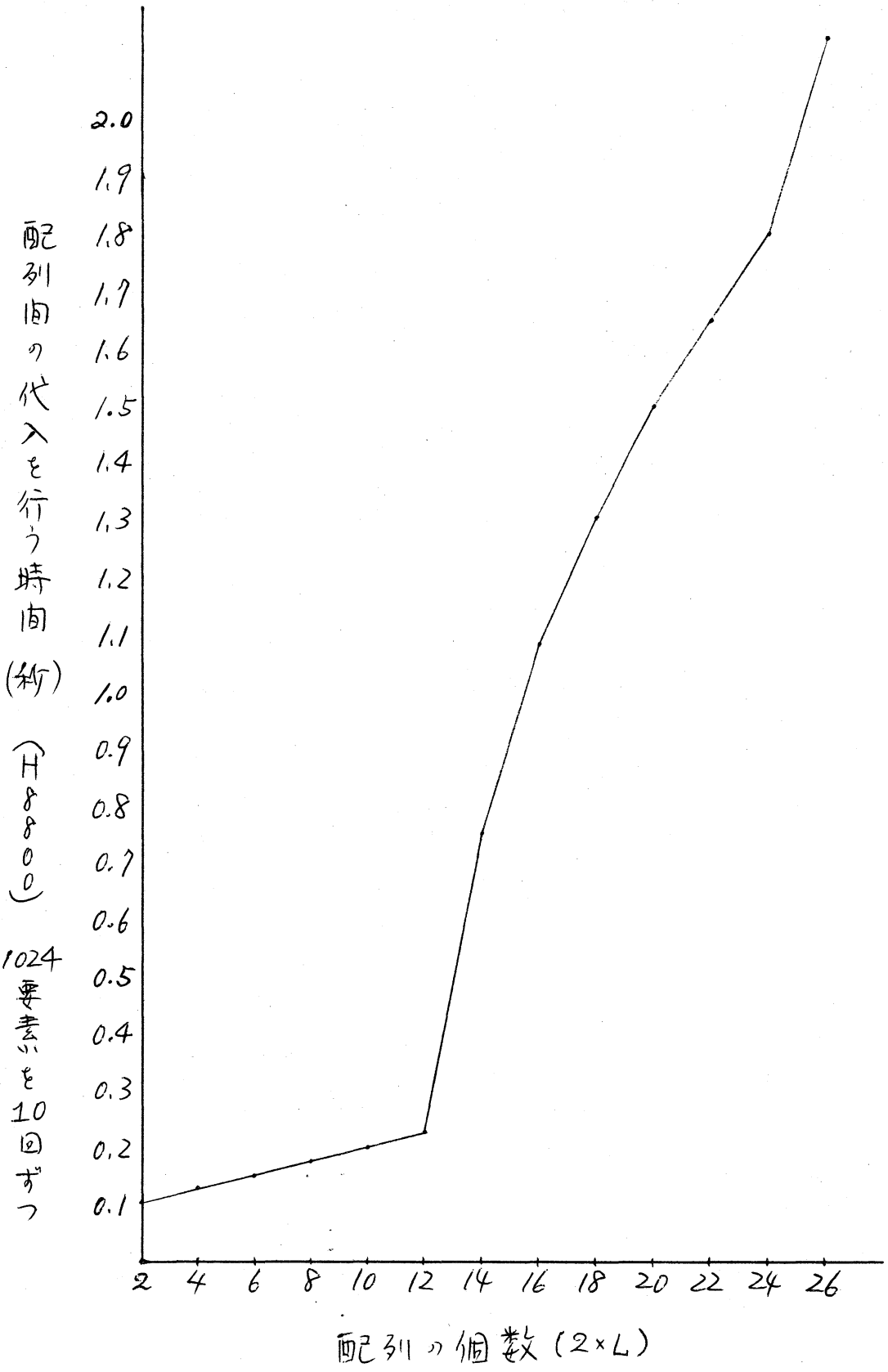


図8　バッファー・メモリの効果

5. パイプライン制御とプログラミング

仮想記憶とは直接関係ないがHITAC 8800ではパイプライン制御（高度先行制御）方式がとられ，図1にも示したように16Bの命令バッファーが使われている．このため

命令バッファ内に入るように

```
→LOOP: LD    0,0(11,8)   ④(バイト) / B(L,J)をFL0へ
       AR    8,4         ② / L+8→L (配列B用)
       MD    0,0(12,7)   ④ / FL0上でA(I,L)をかける
       AR    7,5         ② / L+N*8→L (配列A用)
       ADR   2,0         ② / FL2にFL0をたしこむ
       BCTR  9,10        ② / 終りでなければジャンプ
```

のようなループは15サイクル(約800 n sec)の高速で実行される．

```
C  PRIME NUMBERS
      DIMENSION M(600000)
      M(1)=1
      K=600000
      N=0
      DO 1 I=1,K
      IF(M(I).NE.1) GO TO 1
      DO 2 J=I,K,I
      M(J)=I
    2 CONTINUE
      N=N+1
      M(N)=I
    1 CONTINUE
      WRITE(6,100) N
  100 FORMAT(6H SOSUU,I5)
      L=N-800
      WRITE(6,101)(M(I),I=L,N)
  101 FORMAT(15I7)
      STOP
      END
```

図9　素数探しプログラム ----- 仮想記憶向きでない例
（原プログラムは東大計数工学科の筧氏による）

6. おわりに

図1に示したようなメモリー・ハイラーキーをもつシステムでは，本稿でのべたように処理速度が非常にcontext sensitiveであり，プログラムの性質（ふるまい）に大幅に左右される．したがって従来使われていたプログラムの中には，図9に示したエラトステネスのふるいによる素数探しプログラムのように，大きな実メモリには向いているが，仮想記憶には向かないものもある．実際このプログラムでは，主記憶枠を最大の600KWにとり，DIMENSION M(600000)として600000以下のすべての素数がわずか数秒で求められたのに対し，DIMENSION (700000)としたら15分待っても答の出なかった経験がある．

このことからも想像されるように，従来の小さな（64KW～128KW）実メモリ上で使われていたプログラムや数値計算法のうちには，メモリー・ハイラーキーをもつシステム用に再検討する必要のあるものが多いと思われる．もちろん何ら特別の工夫をしなくても大きなメモリの使えるのが仮想記憶の特徴ではあるが，大きなメモリを〝効率よく〟使うにはそれだけの工夫がいるということである．

この意味から仮想記憶を含むメモリー・ハイラーキー系に関しては今後次のような研究が望まれよう．

(1) 仮想記憶・バッファーメモリに適した数値計算法およびプログラミング技法の開拓.

(2) 従来のライブラリー・プログラムやコンパイラの改良.

(3) 大きなメモリ上でのプログラムのふるまいの実態調査や分析, それにもとづくモデル化

(4) 新しいスワッピング・アルゴリズムの工夫。これには上記(3)の成果が資料となろう.

(5) スワッピング用ハードウエアの改良やドラムのICメモリ化。

(6) クラスタリングなどによるプログラム配置の最適化.

(7) 仮想記憶系の性能の定量的な評価.[これには仮想記憶のおかげでプログラミングの手間が省けるというピープル・パフォーマンスの要素が入るのでなかなか難しい。]

最後に本稿をまとめるに当り, 資料の提供および立ちざまなご教示を頂いた日立製作所の大西勉氏, 大木尚氏, 村田健郎氏らに深く感謝したい。

参考文献

1. 石田: 仮想記憶(1)(2)(3), bit, vol.5, No.11 ~ No.13